Symfony 6 - Manual

Tabla de contenidos

1. Introducción

Tabla de contenidos

- Recursos:
  - https://symfony.com/doc/current/index.html
  - https://jnjsite.com/tutoriales-para-aprender-symfony/

En este manual voy a explicar a partir del Pokedex (una enciclopedia de Pokemons) todo lo que viene incluido dentro del manual oficial de Symfony 6 (el primer enlace de arriba).

- https://www.pokemon.com/es/pokedex

IMPORTANTE Symfony puede instalarse perfectamente en Windows (ver apartado instalación), pero este manual está planteado para usarse en Linux (recomendable Ubuntu/Debian o similar)

En el siguiente apartado incluyo, de forma resumida, todo los comandos necesarios para poder trabajar con symfony clasificados por apartados.

Resumen de comandos

Tabla de contenidos

- Crear proyecto y gestionar el Servidor

```
# Crear proyecto e iniciar Servidor
symfony new /var/www/html/symfony6 --version="6.3.*" --webapp
cd /var/www/html/symfony6
symfony server:start  # O pasar a /var/www/html

# Parar el servidor (En otra pestaña de consola!!)
# symfony server:stop
```

- Instalar dependencias

```
# Instalar las dependencias
composer require --dev symfony/maker-bundle
composer require twig
composer require annotations
composer require symfony/orm-pack
composer require symfony/form
```

- Gestión de Bases de Datos

```
# MODIFICAR .env!
php bin/console doctrine:database:create
```

```
# Crear entidad
php bin/console make:entity

# Crear migración y ejecutarla
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

- Comandos adicionales de trabajo

```
# Crear controlador
php bin/console make:controller
# Visualizar ENDPOINTs
php bin/console debug:router
```

2. Instalación

Tabla de contenidos

- Recurso
  - https://symfony.com/doc/current/setup.html

Para poder instalar Symfony 6 en nuestro equipo necesitamos lo siguiente:

- PHP8
  - Extensiones: Ctype, iconv, PCRE, Session, SimpleXML y Tokenizer;
- MySQL (o algún SGBD similar)
- Composer
- Symfony CLI (ver siguiente apartado)

Instalar Symfony CLI

Tabla de contenidos

```
# Bajamos el instalador
cd ~/Descargas
curl -1sLf 'https://dl.cloudsmith.io/public/symfony/stable/setup.deb.sh' |
sudo -E bash
# Instalamos globalmente
sudo apt install symfony-cli

# Comprobamos
symfony

# Si mas adelante nos sale DEPRECATED
sudo rm /usr/local/bin/symfony
# Y repetimos los pasos de la instalación
```

- Extensiones recomendadas para Visual Studio Code
  - Twig Language 2
  - yaml
  - Prettier

Primer proyecto

Tabla de contenidos

```
# IMPORTANTE: Symfony NO tiene que estar obligatoriamente
# en el directorio de publicación de Apache!
# Estructura del comando
# symfony new <carpeta> --version --webapp
symfony new /var/www/html/symfony6 --version="6.3.*" --webapp

# Nos metemos en el directorio del proyecto e iniciamos el servidor
cd /var/www/html/symfony6
symfony server:start

# Si mas adelante queremos parar el servidor debemos ABRIR OTRA PESTAÑA
# de la consola y poner lo siguiente
### symfony server:stop
```

- Y ya podemos ver en el navegador como quedaría:
  - http://127.0.0.1:8000

Carpetas Básicas

Tabla de contenidos

- bin -> Ejecutables principales del sistema
    - console -> php/bin console...
- config -> Archivos de configuración
    - routes -> Listado de rutas
    - services -> Listado de Servicios creados
- migrations -> creación de migraciones de BBDD
- public -> Páginas publicas
- src -> Recursos del sistema
    - Controller -> Controladores (MVC)
    - Entity -> Entidades (objetos)
    - Repository -> Gestión de consultas
- templates -> plantillas (twig)
- var -> caché de la aplicación y registros (logs)
- vendor -> Dependencias
    - bin -> Ejecutables de dependencias
        - doctrine -> BBDD
        - var-dump-server -> Backup BBDD
        - phpunit -> Test Unitarios
    - Symfony -> núcleo de la aplicación
    - sension -> Maker bundle

Crear repositorio

Tabla de contenidos

- IMPORTANTE: Desde Agosto de 2021 se ha dejado de tener acceso a través de usuario/contraseña a Github (por consola)
- Para poder acceder, tenemos que hacerlo a través de usuario/token.
- Para generar nuestro token debemos hacer lo siguiente:
    1. Pulsamos en nuestra foto > Settings > Developer settings
    2. Personal Access tokens > Tokens (classic) > [Generate New token] > Generate New Token (classic)
    3. En la sección New Personal access token (classic) ponemos un nombre en Note. Ej: mirepo
    4. ☑ Repo y a ser posible TODAS las demás acciones.
    5. Pulsamos en Generate token.
    6. Copiamos el token generado en algún lado (mas tarde lo usaremos), por ejemplo un archivo de texto.
- Desde el primer momento vamos a usar GITHub para crear versiones de nuestros proyectos.

1. Lo primero será crearnos una cuenta en Github: https://github.com/join?ref_cta=Sign+up
2. Lo siguiente, será crearnos el repositorio para el proyecto: a) Nos logamos en Github con nuestros datos. b) Nos vamos a la izquierda de la interfaz donde pone Top Repositories y le damos a `New` c) En Repository name ponemos: symfony6 d) En Description ponemos, por ejemplo: Repositorio para el manual de Symfony6 ☑ Private (¡solo este!) f) Le damos a `Create Repository`

Crear Proyecto

Tabla de contenidos

> NOTA: La variable de sistema $HOMEPROJECTS puede referirse a:
>
> - /var/www/html -> Linux
> - C:\xampp\htdocs -> Windows

1. Nos creamos la aplicación de Symfony (completa):

```
cd $HOMEPROJECTS  #NO tiene que ir en el servidor!
# En cualquier momento podemos ver si lo tenemos todo para instalar una
aplicación Symfony:
symfony check:requirements
# Y procedemos a crear un proyecto completo
symfony new /var/www/html/symfony6 --version="6.3.*" --webapp
```

2. Sincronizamos la carpeta actual con la remota NOTA: Cambiar el user por nuestro usuario:

```
cd $HOMEPROJECTS/symfony6
git remote add origin https://github.com/USER/symfony6.git
git branch -M main
git push -u origin main
touch README.md
git add .
git commit -m "Add README.md"
git push
```

3. Lo siguiente que debemos hacer es meter nuestras credenciales en el equipo (para no tener que estar poniéndolas cada vez que la usemos).

- https://git-scm.com/docs/git-credential-store
- Siguiendo estos pasos, guardaremos nuestras credenciales en un archivo oculto dentro de nuestra carpeta de usuario llamado git-credentials.

```
cd $HOMEPROJECTS/symfony5-manual
touch README2.md
git add .
git commit -m "Add README2.md"
git config credential.helper 'store --file ~/.git-credentials'
git push https://github.com/USER/symfony6.git
Username for 'https://github.com': <user>
Password for 'https://ivanrguez1@github.com': <token>
Everything up-to-date
```

4. Por último, vamos a ver un archivo que es MUY IMPORTANTE: .gitignote

- Con este archivo vamos a omitir la subida de determinados elementos al repositorio.
- Por ejemplo tendremos los siguientes archivos y carpetas:
    - .env.local -> Archivo con las configuraciones de entorno locales
    - /vendor -> Carpeta con el núcleo de symfony y paquetes adicionales
- Dejaremos entonces el gitignore de este modo:

```
# Cache and logs (Symfony2)
/app/cache/*
/app/logs/*
!app/cache/.gitkeep
!app/logs/.gitkeep

# Email spool folder
/app/spool/*

# Cache, session files and logs (Symfony3)
/var/cache/*
/var/logs/*
/var/sessions/*
!var/cache/.gitkeep
!var/logs/.gitkeep
!var/sessions/.gitkeep

# Logs (Symfony4)
/var/log/*
!var/log/.gitkeep

# Parameters
/app/config/parameters.yml
/app/config/parameters.ini

# Managed by Composer
/app/bootstrap.php.cache
/var/bootstrap.php.cache
/bin/*
!bin/console
!bin/symfony_requirements
/vendor/

# Assets and user uploads
/web/bundles/
/web/uploads/

# PHPUnit
/app/phpunit.xml
/phpunit.xml

# Build data
```

```
/build/

# Composer PHAR
/composer.phar

# Backup entities generated with doctrine:generate:entities command
**/Entity/*~

# Embedded web-server pid file
/.web-server-pid
```

Clonar Proyecto

Tabla de contenidos

1. Vamos a descargarnos nuestro proyecto del repositorio.

- Evidentemente, **NO** hay que hacerlo cada vez vayamos a trabajar con nuestro proyecto de Symfony
- El ejemplo que vamos a ver es para, llegado el caso, seguir con el proyecto en otro equipo:

```
cd $HOMEPROJECTS/
# Borramos el contenido de la carpeta symfony6
rm -rf symfony6
git clone https://github.com/ivanrguez1/symfony6.git
# Y ponemos nuestro usuario y contraseña...
```

2. El siguiente paso será descargarnos las dependencias y arrancar el servidor de symfony

- **IMPORTANTE**: Cada vez que bajemos una nueva versión de nuestro repositorio tendremos que descarganos las dependencias

```
cd $HOMEPROJECTS/symfony6
# Descargamos las dependencias
composer install
# Iniciamos el servidor
symfony server:start
```

3. Vemos nuestra página en el navegador (por ejemplo Firefox):

- http://127.0.0.1:8000\

4. Además, en cualquier momento podemos ver las posibles vulnerabilidades de nuestro proyecto:

- Están sacadas de la Base de datos oficial del propio Symfony:

```
cd $HOMEPROJECTS/symfony6
symfony check:security
```

5. Por último tenemos disponible la aplicación de ejemplo oficial junto al manual completo de desarrollo rápido:

- https://symfony.com/doc/5.0/the-fast-track/es/index.html

```
cd $HOMEPROJECTS
symfony new my_project_name --demo
```

3. Crear páginas

Tabla de contenidos

- Recursos:
    - https://symfony.com/doc/current/page_creation.html
    - https://symfony.com/bundles/SymfonyMakerBundle/current/index.html
- Nombre de la rama: tema-03
    - git checkout -b tema-03
- Llegados a este punto tenemos dos opciones:
    - 1. Clonar nuestro proyecto de GitHub y añadir una rama nueva por cada tema
    - 2. Seguir con nuestro proyecto en local y añadir una rama nueva por cada tema
- Vamos a presuponer que iremos trabajando en local y, al final, subiremos los cambios a GitHub
- Para crear las páginas tenemos dos opciones:
    - a) Definir las rutas en el archivo config/routes.yaml, asociandolas a su controlador
    - b) Añadiendo anotaciones a los controladores.
- Veremos ambos casos, automatizando el proceso lo máximo posible.
- Para ambas opciones añadiremos las dependencias y arrancaremos el servidor:

```
cd $HOMEPROJECTS/symfony6
composer require annotations                 # Anotaciones para añadir
rutas al controlador
composer require twig                        # Para usar TWIG en el
Frontend
composer require --dev symfony/maker-bundle  # Para crear controladores (y
mas...) por consola
symfony server:start
```

- Con la instalación de maker-bundle tenemos disponible multitud de comandos para ejecutar
    - Los iremos viendo, poco a poco, mas adelante. Pero el listado está disponible por consola:

```
php bin/console
```

Usar routes.yaml

Tabla de contenidos

1. Debemos abrir otra pestaña de consola y cuando nos pregunte por el nombre, pondremos Aleatorio:
    - IMPORTANTE: OJO a las mayúsculas. Aleatorio tiene la primera en mayúscula.

```
php bin/console make:controller
Choose a name for your controller class (e.g. GrumpyElephantController):
 > Aleatorio

 created: src/Controller/AleatorioController.php
 created: templates/aleatorio/index.html.twig

  Success!
```

2. Editamos el archivo src/Controller/AleatorioController.php
    - En este archivo nos saldrá una anotación #[Route] que usaremos mas tarde. Por ahora lo comentamos:

```
<?php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;

class AleatorioController extends AbstractController
{
  //#[Route('/aleatorio', name: 'aleatorio')]
    public function index(): Response
    {
        $numeroAleatorio = random_int(0, 100);

        return new Response(
            '<html><body>Mi aleatorio: ' . $numeroAleatorio . '</body>
</html>'
        );
    }
}
```

3. Editamos el archivo config/routes.yaml NOTA: podemos añadir tantas URLs (ENDPOINTS) como queramos. Pero **NUNCA** podemos repetir el nombre de cada uno. Eso incluye repetir el MISMO NOMBRE en la anotación dentro del controlador y en el routes.yaml.

```
controllers:
  resource:
    path: ../src/Controller/
```

```
    namespace: App\Controller
  type: attribute

num_aleatorio1:
  path: /aleatorio
  controller: App\Controller\AleatorioController::index

# Si queremos añadir otra ruta (ENDPOINT)
# Copiamos, pegamos, y cambiamos nombre y path.
num_aleatorio2:
  path: /mi-aleatorio
  controller: App\Controller\AleatorioController::index
```

4. Ya solo nos queda probarlo en el navegador:
    - http://localhost:8000/aleatorio
5. Para ver el listado con todas las rutas del sistema, iremos a la consola:

```
php bin/console debug:router
```

Usar Anotaciones

Tabla de contenidos

- Vamos a implementar el mismo ejemplo anterior pero usando anotaciones y el renderizado en una página **TWIG**:

> NOTA: Usar la anotación #[Route] hace innecesario la importación de la libreria Route (use Symfony\Component\Routing\Annotation\Route)

1. Usamos el controlador del paso anterior: AleatorioController.php NOTA: Podemos agregar varias páginas al mismo controlador. Ya lo veremos.

2. Editamos la página src/Controller/AleatorioController.php

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class AleatorioController extends AbstractController
{
    // public function index(): Response
    // ...

    // En la anotación ponemos la ruta: http://localhost:8000/aleatorio
    #[Route('/aleatorio2', name: 'aleatorio2')]
    public function index2(): Response
    {
        $numeroAleatorio = random_int(0, 100);

        // Aquí usaremos Twig, y le pasamos el parámetro numeroAleatorio
        // ATENCIÓN: aleatorio se recogerá en el twig entre llaves: {{
aleatorio }}
        return $this->render('aleatorio/index.html.twig', [
            'controller_name' => 'AleatorioController',
            'aleatorio' => $numeroAleatorio,
        ]);
    }
}
```

3. Como hemos visto en el código, vamos a usar Twig, un motor de plantillas.
    - Dicho motor está mantenido por Fabien Potencier, el creador de Symfony
    - Mas información aqui: https://twig.symfony.com/
    - Instalamos el plugin Twig Language (5estrellas)
    - Editamos la plantilla: templates/aleatorio/index.html.twig

- IMPORTANTE: la variable {{ aleatorio }} se corresponde con la clave 'aleatorio' => $numeroAleatorio del controlador.

```
{% extends 'base.html.twig' %}

{% block title %}Hola AleatorioController!{% endblock %}

{% block body %}
<style>
    .example-wrapper { margin: 1em auto; max-width: 800px; width: 95%;
font: 18px/1.5 sans-serif; }
    .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
</style>

<div class="example-wrapper">
    <h1>Hola {{ controller_name }}! ✅</h1>

    <!-- El resto del código lo dejamos o lo borramos...-->
    <p> Tu número aleatorio es: {{ aleatorio }} </p>
</div>
{% endblock %}
```

4. Como vimos antes, podemos probarlo en el navegador:

- http://localhost:8000/aleatorio2

5. Por último, vamos a añadir una tercera ruta, en el mismo controlador:

- En este caso, pasamos un parámetro para que nos devuelva varios aleatorios:
- En la ruta pondremos entre llaves los parámetros que queramos definir:
  - /aleatorio3/{num}
  - Dicho parámetro (num) se pasará a la función como una variable **CON EL MISMO NOMBRE**: $num

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class AleatorioController extends AbstractController
{
    // public function index(): Response
    // public function index2(): Response
    // ...

    #[Route('/aleatorio3/{num}', name: 'aleatorio3')]
    public function index3(int $num): Response
```

```
    {
        $numerosAleatorios = "<br>";
        for ($i = 1; $i <= $num; $i++) {
            $numerosAleatorios .= random_int(0, 100) . "<br>";
        }
        return new Response(
            '<html><body>Números aleatorios: ' . $numerosAleatorios .
'</body></html>'
        );
    }
}
```

6. Como vimos antes, podemos probarlo en el navegador:

- http://localhost:8000/aleatorio3/10
    - Nos sacaria 10 aleatorios

7. ¿Y si queremos mas de un parámetros?

- Pues igual, metemos /{param1}/{param2}...
- En este caso, pasamos dos parámetros para que nos devuelva varios aleatorios con un límite. Eso si, esta vez vamos a usar el renderizado de la plantilla TWIG:

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class AleatorioController extends AbstractController
{
    // public function index(): Response
    // public function index2(): Response
    // public function index3(): Response
    // ...

    #[Route('/aleatorio4/{num1}/{limite}', name: 'aleatorio4')]
    public function index4(int $num): Response
    {
        $numeroAleatorio = "";
        for ($i = 1; $i <= $num1; $i++) {
            $numeroAleatorio .= random_int(0, $limite) . "-";
        }
        return $this->render('aleatorio/index.html.twig', [
            'controller_name' => 'AleatorioController',
            'aleatorio' => $numeroAleatorio,
        ]);
    }
}
```

8. Como vimos antes, podemos probarlo en el navegador:

- http://localhost:8000/aleatorio4/5/20
    - Nos sacaria 5 aleatorios entre el 0 y el 20.

9. Entre las muchas opciones que nos da el uso de rutas y controladores está el emplear clases adicionales dentro del controlador y pasar, por ejemplo, objetos al twig. Un ejemplo:

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class Camion
{
    // Atributos
    public $modelo;
    public int $potencia;

    // Constructor
    public function __construct()
    {
        $this->modelo = "Volvo FH Electric";
        $this->potencia = 240;
    }
    // toString
    public function __toString(): string
    {
        $mensaje = "";
        $mensaje .= $this->modelo . "<br>";
        $mensaje .= $this->potencia . "<br>";
        return $mensaje;
    }
}

class AleatorioController extends AbstractController
{
    //...

    #[Route('/aleatorios', name: 'aleatorio5')]
    public function aleatorios(): Response
    {
        $numeroAleatorios = array();
        for ($i = 0; $i < 5; $i++) {
            $numeroAleatorios[] = random_int(0, 100);
        }

        // Me creo un camion
```

```
        $camion = new Camion();

        // El método implode convierte Array -> String
        //return new Response(implode(",", $numeroAleatorios));

        // Para pruebas
        // return new Response($camion);

        return $this->render('aleatorio/index2.html.twig', [
            'controller_name' => 'AleatorioController',
            "numAleatorios" => $numeroAleatorios,
            "camion" => $camion,
        ]);
    }
}
```

- El twig, finalmente, quedaría así:
  - En /templates/aleatorio/index2.html.twig

```
{% extends 'base.html.twig' %}

{# Bloque titulo #}
{% block title %}
  Ejemplo Aleatorios
{% endblock %}

{% block stylesheets %}
 <style>
 body {
    background: black;
    color: white;
  }

  .example-wrapper {
      margin: 1em auto;
      max-width: 800px;
      width: 95%;
      font: 18px/1.5 sans-serif;
    }
    .example-wrapper code {
      background: #f5f5f5;
      padding: 2px 6px;
    }
 </style>

{% endblock %}

{% block body %}
  <div class="example-wrapper">
    <h1 class="text-info">Hello {{ controller_name }}! ✅</h1>
```

```
    <p class="text-danger">Los números aleatorios son:</p>

  {% for aleatorio in numAleatorios %}
    {{ aleatorio }} <br />
  {% endfor %}

  <h2> Mi Camion!</h2>
  {{ camion.modelo }} <br>
  {{ camion.potencia }} <br>

<h2> Mi Camion Entero!</h2>
{{ camion|raw }} <br>
    This friendly message is coming from:<ul>
      <li>
        Your controller at <code><a href="{{
'/var/www/html/Symfony/src/Controller/AleatorioController.php'|file_link(0)
}}">src/Controller/AleatorioController.php</a></code>
      </li>
      <li>
        Your template at <code><a href="{{
'/var/www/html/Symfony/templates/aleatorio/index.html.twig'|file_link(0)
}}">templates/aleatorio/index.html.twig</a></code>
      </li>
    </ul>
  </div>
{% endblock %}
```

10. Una vez finalizado todo el trabajo, es hora de subirlo todo al repositorio:

> NOTA: Hay que recordar que en usuario pondremos el nuestro, pero en password, pondremos nuestro TOKEN

```
git add .
git commit -m "Tema-03 Crear Paginas"
git push --set-upstream origin tema-03
git push
```

4. Doctrine

Tabla de contenidos

- Symfony proporciona todas las herramientas necesarias para usar bases de datos en las aplicaciones gracias a Doctrine , el mejor conjunto de bibliotecas PHP para trabajar con bases de datos.
- Estas herramientas admiten bases de datos relacionales como MySQL y PostgreSQL (o SQLite, que será el que usemos) y también bases de datos NoSQL como MongoDB.
- Recursos:
    - https://symfony.com/doc/current/doctrine.html
    - https://symfony.com/doc/current/doctrine/dbal.html

Instalación de Doctrine

Tabla de contenidos

1. Creamos la rama y nos introducimos en ella

```
git checkout -b tema-04
git push --set-upstream origin tema-04
```

2. Por consola, dentro del proyecto:

```
cd $HOMEPROJECTS/symfony6
composer require symfony/orm-pack
composer require --dev symfony/maker-bundle
```

3. Ahora toca configurar la Base de datos que vamos a usar. Para ello tenemos el archivo .env.

- MUY IMPORTANTE: Si queremos cambiar la configuración de trabajo en local, debemos crearnos un archivo llamado .env.local que tendrá la configuración que deseemos.
- Para nuestro caso, los cambios los haremos en el .env, trabajando siempre con MySQL.
- Como ya veremos en el código, tan solo debemos cambiar una linea para trabajar, por ejemplo, con SQLite
- Una ultima cosa: en la línea sin comentar (DATABASE_URL="mysql...) debemos poner el nombre de la base de datos que queramos usar. Por defecto es db_name, pero nosotros pondremos symfony6
- El formato para mysql es el siguiente:
    - mysql://root:root@127.0.0.1:3306/test
        - Usuario:root; clave: root; ip_local: 127.0.0.1; puerto: 3306; Base Datos: test
    - sgbd://usuario:clave@ip_local:puerto/nombre_bbdd

```
###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=72984f7cb2d89365e92bb4dd28f0c02f
```

```
###< symfony/framework-bundle ###


###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-
dbal/en/latest/reference/configuration.html#connecting-using-a-url
# IMPORTANT: You MUST configure your server version, either here or in
config/packages/doctrine.yaml
#
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
# DATABASE_URL="mysql://root:root@127.0.0.1:3306/test?
serverVersion=10.2.32-MariaDB-log"

#DATABASE_URL="mysql://root:root@127.0.0.1:3306/db_name?serverVersion=5.7"
# CAMBIAR por la configuración correcta de MySQL!
DATABASE_URL="mysql://root:root@127.0.0.1:3306/pokedex?
serverVersion=8&charset=utf8mb4"

# DATABASE_URL="postgresql://db_user:db_password@127.0.0.1:5432/db_name?
serverVersion=13&charset=utf8"
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
###< doctrine/doctrine-bundle ###
```

- Para los que tengan Raspberry

```
DATABASE_URL="mysql://root:root@127.0.0.1:3306/pokedex"
```

Pokedex

Tabla de contenidos

- El Pokedex es la Base de datos de los Pokemons
    - https://www.pokemon.com/es/pokedex
- En este manual vamos a crearnos la estructura básica para guardar pokemons, con 3 tablas, que debemos meter por este orden:
- 1. Tipos (Tabla Principal)
    - https://www.wikidex.net/wiki/Tipo

Tipos

idTipo	INT(AUTO)	NOT NULL
tipo	VARCHAR(45)	NOT NULL

- 2. Categorias (Tabla Principal)
    - https://www.wikidex.net/wiki/Categoría

Categorias

idCategoria	INT(AUTO)	NOT NULL
categoria	VARCHAR(45)	NOT NULL

- 3. Pokemons (tabla derivada)

Pokemons

idPokemon	INT	NOT NULL
nombre	VARCHAR(45)	NOT NULL
sexo	BOOL	NULL
sexo2	BOOL	NULL
idCategoria	INT	NOT NULL
idTipo	INT	NOT NULL
idTipo2	INT	NULL
evolucion	BOOL	NOT NULL

- Consideraciones importantes:
    - Las IDs de las categorias y tipos son automáticas
    - Las IDs de los Pokemons son especificas (están numerados)
    - Una raza de Pokemon puede tener 0/1/2 sexos
    - Una raza de Pokemon puede tener 1/2 tipos

Estructura de la BD

Tabla de contenidos

1. Ahora nos creamos la base de datos:

> IMPORTANTE: En vez de db_name debe estar puesta la cadena con symfony6 o el nombre que queramos

```
php bin/console doctrine:database:create
> Created database `pokedex` for connection named default
# Para ver todos los comandos disponibles de Doctrine
php bin/console list doctrine
```

2. Ahora vamos a crearnos una tabla. En Doctrine se le llama Entity (entidad) y no es mas que una clase que incluye atributos (campos) y sus correspondientes métodos setter y getter.

> Por convencion, las entidades empiezan por mayúsculas y los campos van en minúsculas

> Al crear las entidades, no metemos como campo las IDs. Estas se ponen de forma automática. De todos modos, si tenemos que modificar algo, siempre podemos hacerlo "a mano" mas adelante (como veremos).

```
php bin/console make:entity
> Tipos

 created: src/Entity/Tipos.php
 created: src/Repository/TiposRepository.php

 Entity generated! Now let's add some fields!
 You can always add more fields later manually or by re-running this
command.

New property name (press <return> to stop adding fields):
> tipo

Field type (enter ? to see all types) [string]:
> string

Field length [255]:
> 255

Can this field be null in the database (nullable) (yes/no) [no]:
> no

updated: src/Entity/Tipos.php

Add another property? Enter the property name (or press <return> to stop
adding fields):
>
```

```
# INTRO!

Success!

Next: When you're ready, create a migration with php bin/console
make:migration
```

3. Vale, ya tenemos la primera tabla, ahora vamos por la segunda:

- En este caso será la tabla Categorias

```
php bin/console make:entity
Class name of the entity to create or update (e.g. OrangeKangaroo):
 > Categorias

 created: src/Entity/Categorias.php
 created: src/Repository/CategoriasRepository.php

 Entity generated! Now let's add some fields!
 You can always add more fields later manually or by re-running this
command.

 New property name (press <return> to stop adding fields):
 > categoria

 Field type (enter ? to see all types) [string]:
 >

 Field length [255]:
 >

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Categorias.php

 Add another property? Enter the property name (or press <return> to stop
adding fields):
 >

  # INTRO!

  Success!
```

4. Vale, ya tenemos las dos primeras tablas, ahora vamos por la tercera

- En este caso será la tabla Pokemons, que se relacionará con las anteriores

```
php bin/console make:entity
Class name of the entity to create or update (e.g. OrangeKangaroo):
```

```
 > Pokemons

 created: src/Entity/Pokemons.php
 created: src/Repository/PokemonsRepository.php

 Entity generated! Now let's add some fields!
 You can always add more fields later manually or by re-running this
command.

 New property name (press <return> to stop adding fields):
 > categoria

 Field type (enter ? to see all types) [string]:
 >

 Field length [255]:
 >

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Categorias.php

 Add another property? Enter the property name (or press <return> to stop
adding fields):
 >

  # INTRO!

  Success!
```

5. Evidentemente, en cualquier momento podemos añadir campos adicionales a la tabla:

- Para ello sólo debemos ejecutar el comando de creación de la entidad con el MISMO NOMBRE...

> Vamos a ver con ? TODAS las opciones de campo disponibles...

```
php bin/console make:entity
> Pokemons

Your entity already exists! So let's add some new fields!

New property name (press <return> to stop adding fields):
 > sexo

 Field type (enter ? to see all types) [string]:
 > ?

Main types
  * string
  * text
  * boolean
```

```
  * integer (or smallint, bigint)
  * float

Relationships / Associations
  * relation (a wizard 🧙 will help you build the relation)
  * ManyToOne
  * OneToMany
  * ManyToMany
  * OneToOne

Array/Object Types
  * array (or simple_array)
  * json
  * object
  * binary
  * blob

Date/Time Types
  * datetime (or datetime_immutable)
  * datetimetz (or datetimetz_immutable)
  * date (or date_immutable)
  * time (or time_immutable)
  * dateinterval

Other Types
  * ascii_string
  * decimal
  * guid
  * json_array

Field type (enter ? to see all types) [string]:
 > boolean

 Can this field be null in the database (nullable) (yes/no) [no]:
 > yes

 updated: src/Entity/Pokemons.php

 Add another property? Enter the property name (or press <return> to stop
adding fields):
 >
# INTRO!!

  Success!
```

6. Seguimos el proceso con sexo2 y evolucion. El siguiente paso será relacionar esta tabla con las demás.

- Tipos y Categorias serán las tablas principales y Pokemons la tabla derivada
- Por tanto, para crear la relación NOS VAMOS A LA DERIVADA (Pokemons)
- El tipo será ManyToOne (muchos Pokemons son de 1 Tipo)
- El nombre a poner es el mismo de la entidad, PERO EN MINUSCULAS!

- Mas información: https://symfony.com/doc/current/doctrine/associations.html

```
php bin/console make:entity
Class name of the entity to create or update (e.g. DeliciousChef):
 > Pokemons

 Your entity already exists! So let's add some new fields!

 New property name (press <return> to stop adding fields):
 > categoria

Field type (enter ? to see all types) [string]:
 > ManyToOne

 What class should this entity be related to?:
 > Categorias

 Is the Pokemons.categoria property allowed to be null (nullable)? (yes/no)
[yes]:
 > no

 Do you want to add a new property to Categorias so that you can
access/update Pokemons objects from it - e.g. $categorias->getPokemons()?
(yes/no) [yes]:
 >

 A new property will also be added to the Categorias class so that you can
access the related Pokemons objects from it.

 New field name inside Categorias [pokemons]:
 >

 Do you want to activate orphanRemoval on your relationship?
 A Pokemons is "orphaned" when it is removed from its related Categorias.
 e.g. $categorias->removePokemons($pokemons)

 NOTE: If a Pokemons may *change* from one Categorias to another, answer
"no".

 Do you want to automatically delete orphaned App\Entity\Pokemons objects
(orphanRemoval)? (yes/no) [no]:
 >

 updated: src/Entity/Pokemons.php
 updated: src/Entity/Categorias.php

 Add another property? Enter the property name (or press <return> to stop
adding fields):
 >

  Success!
```

```
 Next: When you're ready, create a migration with php bin/console
make:migration
```

7. Seguimos el mismo proceso con Tipo, aunque en este caso ambos campos pueden ser nulos.
8. Por supuesto, en cualquier momento podemos modificar algún dato en la propia entidad: src/Entity/Pokemons.
9. Por último, toda la lógica dentro de Symfony debemos trasladarla al SGBD:

```
php bin/console make:migration
Success!

 Next: Review the new migration "migrations/Version20230615095503.php"
```

- Como nos dice la consola podemos ver el archivo migrations/Version20230615095503.php" para personalizar los comandos SQL que se van a ejecutar respecto al SGBD.
    - Por experiencia, suele ser buena idea cambiar los nombres de los FK (Foreign Key) e IDX (Index)
- Y lo trasladamos:

```
php bin/console doctrine:migrations:migrate

WARNING! You are about to execute a migration in database "symfony5" that
could result in schema changes and data loss. Are you sure you wish to
continue? (yes/no) [yes]:
 >
  # INTRO!

[notice] Migrating up to DoctrineMigrations\Version20220615185503
[notice] finished in 57.7ms, used 14M memory, 1 migrations executed, 3 sql
queries
```

- Para visualizar TODOS los comandos disponibles de doctrine:

```
php bin/console doctrine
```

- Podemos comprobarlo todo en MySQL (abrimos sensión):

```
USE Pokedex;
SHOW TABLES;
DESCRIBE tipos;
DESCRIBE categorias;
DESCRIBE pokemons;
```

Persistir Objetos

Tabla de contenidos

En este apartado vamos a insertar datos en las 3 tablas. Por supuesto debemos comenzar por las tablas principales, por ejemplo con tipos (son 18).

- https://www.wikidex.net/wiki/Lista_de_Pokémon_por_tipo

De forma resumida veremos que los datos en los endpoints se pueden meter de 3 formas distintas:

- a. Registro por registro en el controlador
- b. Con un array de registros en el controlador
- c. Mediante parámetros para cada campo (lo recomendado)

1. Para ello podemos crearnos un Controlador (también podemos hacerlo con un servicio):

```
# Ahora usamos
# php bin/console make:controller <nombre_controlador>
php bin/console make:controller TiposController
```

2. Dentro de dicho controlador ponemos toda la lógica para insertar un registro:

- En src/Controller/TiposController

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

// Importamos la libreria para insertar con Doctrine (Gestor de Registros)
use Doctrine\Persistence\ManagerRegistry;
use App\Entity\Tipos;
use App\Repository\TiposRepository;

class TiposController extends AbstractController
{
  #[Route('/insertar-tipo', name: 'tipos1')]
  public function insertar1Tipo(ManagerRegistry $doctrine): Response
  {
      // Creamos el objeto Gestor de Entidad
      $entityManager = $doctrine->getManager();

      // Defino un objeto tipo
      $tipo = new Tipos();
      $tipo->setTipo("Acero");
```

```
        // Y lo guardo
        $entityManager->persist($tipo);
        $entityManager->flush();

        // Usamos el método getId para saber el nuevo ID guardado
        return new Response('Guardado Tipo con ID -> ' . $tipo->getId());
    }
}
```

- Lo probamos en el navegador
    - http://127.0.0.1:8000/insertar-tipo

4. También podemos meter varios tipos a la vez con un array:

- En src/Controller/TiposController

```
<?php
namespace App\Controller;

// ...

class TiposController extends AbstractController
{
    // ...

    // OJO! La ruta y su name deben ser distintos al método anterior
    #[Route('/insertar-tipos', name: 'tipos2')]
    public function insertarVariosTipos(ManagerRegistry $doctrine): Response
    {
        // Creamos el objeto Gestor de Entidad
        $entityManager = $doctrine->getManager();

        // En el caso de tener mas campos, tan solo debemos añadirlos en cada
        // array interno: tipo1, tipo2, tipo3
        $registros = array(
            "tipo1" => array(
                "tipo" => 'Agua',
            ),
            "tipo2" => array(
                "tipo" => 'Bicho',
            ),
            "tipo3" => array(
                "tipo" => 'Dragón',
            ),
        );

        // Ahora empleamos un foreach para guardar tipo por tipo
        // Igual que lo hicimos en el método anterior
        foreach ($registros as $registro) {
            $tipo = new Tipos();
            $tipo->setTipo($registro['tipo']);
```

```
            $entityManager->persist($tipo);
            $entityManager->flush();
        }

        // Aquí directamente ponemos un mensaje
        return new Response('Guardados Tipos!');
    }
}
```

- Y lo probamos en el navegador
    - http://127.0.0.1:8000/insertar-tipos

5. Por último, vamos a ver como meter datos con parámetros en el endpoint:

- Debemos incluir cada parámetro entre llaves en el endpoint
    - /insertar-tipo/{valorTipo}
- Dicho parámetro se añade en la declaración de la función, !siempre con el mismo nombre!
    - insertarTipo(ManagerRegistry $doctrine, string $valorTipo)
    - El resto se hace igual que hemos visto anteriormente.
- En src/Controller/TiposController

```
namespace App\Controller;

// ...

class TiposController extends AbstractController
{
    // ...
    #[Route('/insertar-tipo/{valorTipo}', name: 'tipos3')]
    public function insertarTipo(ManagerRegistry $doctrine,
    string $valorTipo): Response
    {
        $entityManager = $doctrine->getManager();
        $tipo = new Tipos();
        $tipo->setTipo($valorTipo);
        $entityManager->persist($tipo);
        $entityManager->flush();
        $mensaje = 'Guardado Tipo:  ' . $valorTipo;

        return new Response($mensaje);
    }
}
```

6. Repetimos el mismo proceso para Categorias:

- En src/Controller/CategoriasController

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

// Importamos la libreria para insertar con Doctrine (Gestor de Registros)
use Doctrine\Persistence\ManagerRegistry;
use App\Entity\Categorias;

class CategoriasController extends AbstractController
{
    #[Route('/insertar-categoria/{valorCategoria}', name: 'categoria1')]
    public function insertarCategoria(ManagerRegistry $doctrine,
      string $valorCategoria): Response
    {
        $entityManager = $doctrine->getManager();
        $categoria = new Categorias();
        $categoria->setCategoria($valorCategoria);
        $entityManager->persist($categoria);
        $entityManager->flush();
        $mensaje = 'Guardado categoria:  ' . $valorCategoria;

        return new Response($mensaje);
    }
}
```

7. Ahora toca ponerse con la tabla relacionada. Para este caso debemos tener varias cosas en cuenta:

- Debemos añadir los repositorios y entidades de las 3 tablas
- Para las claves foráneas emplearemos sus correspondientes IDs (a la larga será lo que empleemos por ejemplo en los formularios)
- Para esas claves, emplearemos el método find del repositorio de la entidad con la que se relaciona.
- Para el booleano sexo/sexo2: 0 -> Sin asignar; 1-> Hembra; 2-> Macho

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Doctrine\Persistence\ManagerRegistry;
use App\Entity\Tipos;
use App\Entity\Categorias;
use App\Entity\Pokemons;

// Vamos a importar los 3 repositorios de las 3 tablas
```

```
use App\Repository\CategoriasRepository;
use App\Repository\PokemonsRepository;
use App\Repository\TiposRepository;

class PokemonsController extends AbstractController
{
  // Para probar:
  // http://127.0.0.1:8000/insertar-pokemon/4/Charmander/1/2/2/7/0/1
  // http://127.0.0.1:8000/insertar-pokemon/5/Charmaleon/1/2/3/7/0/1
  // http://127.0.0.1:8000/insertar-pokemon/132/Ditto/0/0/4/8/0/0
  #[Route('/insertar-
pokemon/{id}/{nombre}/{sexo}/{sexo2}/{cat}/{tipo1}/{tipo2}/{evolucion}',
    name: 'pokemon1')]
  public function index(
      ManagerRegistry $doctrine,
      int $id,
      string $nombre,
      int $sexo,
      int $sexo2,
      int $cat,
      int $tipo1,
      int $tipo2,
      int $evolucion
  ): Response {
      // La primera parte es similar a lo ya visto anteriormente...
      $entityManager = $doctrine->getManager();
      $pokemon = new Pokemons();
      $pokemon->setId($id);
      $pokemon->setNombre($nombre);

      // Así se hacen las asignaciones de booleanos
      // ATENCIÓN: Las asignaciones para el sexo serán:
      // 0 -> Sin asignar; 1-> Hembra; 2-> Macho
      if ($sexo != 0) {
          $pokemon->setSexo($sexo);
      }
      if ($sexo2 != 0) {
          $pokemon->setSexo2($sexo2);
      }
      $pokemon->setEvolucion($evolucion);

      // Y ahora para poner las IDs de claves foráneas, usamos el
      // repositorio de cada entidad
      $idCategoria = $entityManager->getRepository(Categorias::class)-
>find($cat);
      $pokemon->setIdCategoria($idCategoria);
      $idTipo = $entityManager->getRepository(Tipos::class)->find($tipo1);
      $pokemon->setIdTipo($idTipo);
      if ($tipo2 != 0) {
          $idTipo2 = $entityManager->getRepository(Tipos::class)-
>find($tipo2);
          $pokemon->setIdTipo2($idTipo2);
      }
```

```
        $entityManager->persist($pokemon);
        $entityManager->flush();

        $mensaje = 'Guardado Pokemon:  ' . $nombre;
        return new Response($mensaje);
    }
}
```

8. Y si queremos, lo comprobamos directamente con estos comandos de Symfony:

```
php bin/console dbal:run-sql 'SELECT * FROM tipos'
php bin/console dbal:run-sql 'SELECT * FROM categorias'
php bin/console dbal:run-sql 'SELECT * FROM pokemons'
```

9. Incluso, si lo deseamos, podemos ejecutar un JOIN con todas las tablas:
    - Para cada campo se ha usado un alias (As...) que cambia el nombre de las columnas (incluso con tildes)
    - Para las columnas de sexo se ha usado un CASE WHEN ... THEN ... ELSE ... END, que escribe valores en función del digito guardado
    - El JOIN se ha hecho con categoria y el primer campo de Tipo
    - El segundo campo Tipo NO es necesario sacarlo (por ahora)

```
php bin/console dbal:run-sql "SELECT nombre As Nombre,
    CASE WHEN sexo = '1' THEN 'Hembra'
        WHEN sexo = '2' THEN 'Macho'
        ELSE '--' END AS Sexo,
    CASE WHEN sexo2 = '1' THEN 'Hembra'
        WHEN sexo2 = '2' THEN 'Macho'
        ELSE '--' END AS Sexo2,
    CASE WHEN evolucion = '1' THEN 'Si'
        ELSE 'No' END AS Evolución,
    categoria as Categoria, tipo as Tipo
FROM pokemons, tipos, categorias
WHERE id_categoria_id = categorias.id
AND id_tipo_id = tipos.id"
```

Consultar Objetos

Tabla de contenidos

Una vez visto como INSERTAR objetos en la BBDD (persistir), toca ver las consultas (la R del CRUD). Para este caso veremos distintos tipos de consultas que se dividen en los siguientes:

- a. Consulta general de la tabla (findAll) con salida en formato Tabla (array asociativo)
- b. Consulta general de la tabla (findAll) con renderizado TWIG
- c. Consulta General de la tabla (findAll) con salida JSON (array asociativo)
- d. Consulta por parámetro (findBy) con salida JSON (array asociativo)
- e. Consulta por criterio (findOneBy, un registro) o por ID (find, un registro) con salida JSON.
- f. Consulta por SQL (fetchAllAssociative), salida JSON.

Además de los anteriores (que son los mas comunes) que ya vienen definidos en el repositorio correspondiente de la entidad podemos construir nuestros propios métodos y usarlos (como ya veremos por ejemplo en actualizar)

1. Consulta general de la tabla Categorias con salida HTML (table)

- Lo haremos sobre el controlador de Categorias
- No lleva parámetros. Usaremos el método del repositorio findAll
- En src/Controller/CategoriasController.php

```

```

2.

ME HE QUEDADO AQUI!!

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
//...

class CategoriasController extends AbstractController
{
    #[Route('/insertar-categoria/{valorCategoria}', name: 'categoria1')]
    public function insertarCategoria(ManagerRegistry $doctrine, string
$valorCategoria): Response
    //...

    #[Route('/consultar-categorias', name: 'categoria2')]
```

```
    public function consultarCategorias(ManagerRegistry $doctrine):
Response
    {
        // Podemos usar tanto $doctrine como el repositorio de Categorias
        $repoCategorias = $doctrine->getRepository(Categorias::class);
        $categorias = $repoCategorias->findAll();
        // Hay 3 opciones para "pintar" los registros
        // 1. Response (table)
        // 2. JSON (es el mas usado)
        // 3. Render (Twig)

        $tabla = "<table border='1'>";
        $tabla .= "<tr>";
        $tabla .= "<td>Id</td>";
        $tabla .= "<td>Categoria</td>";
        $tabla .= "</tr>";
        // OJO! Es un array de OBJETOS!!
        // Hay que usar los getter!
        foreach ($categorias as $categoria) {
            $tabla .= "<tr>";
            $tabla .= "<td>" . $categoria->getId() . "</td>";
            $tabla .= "<td>" . $categoria->getCategoria() . "</td>";
            $tabla .= "</tr>";
        }
        $tabla .= "</table>";

        return new Response($tabla);
    }
}
```

8. Inserción de datos por parámetros: en este caso, vamos pasar los 3 campos por el ENDPOINT. ATENCIÓN: en este caso debemos insertar el autor como un objeto, buscando primero su registro por la ID pasada por el parámetro y asignado luego su valor en el registro de articulos.

Nos creamos un nuevo método crearArticulo:

- En src/Controller/ArticulosController

```
#[Route('/crea-articulo/{titulo}/{publicado}/{autor}', name: 'crea-
articulo')]
    public function crearArticulo(
        ManagerRegistry $doctrine,
        String $titulo,
        int $publicado,
        int $autor
    ): Response {
        $entityManager = $doctrine->getManager();

        $articulo = new Articulos();
        $articulo->setTitulo($titulo);
        $articulo->setPublicado($publicado);
```

```
        // Para el caso del autor, debemos buscar el autor
        // con la ID pasada por parámetro
        $autor = $entityManager->getRepository(Autores::class)-
>find($autor);
        $articulo->setAutor($autor);

        $entityManager->persist($articulo);
        $entityManager->flush();

        return new Response('Articulo agregado');
    }
```

Consultar Objetos

Tabla de contenidos

- Ya hemos visto como realizar el INSERT en el apartado anterior.
- A partir de ahora vamos a realizar el resto de componentes del CRUD, comenzando por el SELECT

1. Para empezar sacaremos el SELECT * FROM articulos usando como salida una tabla HTML usando el findAll():

- En src/Controller/ArticulosController

```
// Añadimos el repositorio a las clases que usamos:
use App\Repository\ArticulosRepository;
```

- Vamos a crearnos un nuevo método en el controlador de Articulos:

```
  #[Route('/ver-articulos', name: 'ver-articulos')]
  public function mostrarArticulos(ArticulosRepository $repo): Response
  {
      $articulos = $repo->findAll();
      $respuesta = "<html>
      <body>
          <table border=1>
              <th>ID</th>
              <th>Titulo</th>
              <th>publicado</th>
              <th>autor</th>";
      // con getAutor obtenemos el autor como objeto
      // Con eso, sacamos lo que queramos, por ejemplo el nombre
      foreach ($articulos as $articulo) {
          $respuesta .= "<tr>
          <td> " . $articulo->getId() . "</td>
          <td> " . $articulo->getTitulo() . "</td>
          <td> " . $articulo->isPublicado() . "</td>
          <td> " . $articulo->getAutor()->getNombre() . "</td>
          </tr>";
      }
      $respuesta .= "</table>
      </body>
      </html>";
      return new Response($respuesta);
  }
```

- Lo podemos ver en el navegador
  - http://localhost:8000/ver-articulos

2. Ahora vamos a sacar SELECT * FROM articulos WHERE id = 1 usando como salida JSON, usando el find($id):

- Creamos otro método en el mismo controlador:
  - En src/Controller/ArticulosController

```
// Añadimos el JsonResponse para sacarlo en formato JSON (Para API REST)
use Symfony\Component\HttpFoundation\JsonResponse;
```

- Y creamos el nuevo método:

```
#[Route('/articulo/{id}', name: 'ver-articulo')]
public function verArticulo(ManagerRegistry $doctrine, int $id): Response
{
    $articulo = $doctrine->getRepository(Articulos::class)->find($id);
    // De nuevo, sacamos el autor con el objeto completo...
    return new JsonResponse([
        'id' => $articulo->getId(),
        'titulo' => $articulo->getTitulo(),
        'publicado' => $articulo->isPublicado(),
        'autor' => $articulo->getAutor()->getNombre(),
    ]);
}
```

- Lo visualizamos en el navegador, usando el JsonViewer de Chrome:
  - http://localhost:8000/articulo/1
- https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh?hl=es

3. Vamos a sacar SELECT * FROM articulos usando como salida JSON, usando el findAll():

- Creamos otro método en el mismo controlador:
  - En src/Controller/ArticulosController
- Debemos tener estas clases al principio del controlador:

```
namespace App\Controller;

use App\Entity\Articulos;
use App\Entity\Autores;
use Doctrine\Persistence\ManagerRegistry;
use Symfony\Component\HttpFoundation\JsonResponse;
use App\Repository\ArticulosRepository;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
```

- Añadimos un nuevo método con su ruta:

```
#[Route('/consultar-articulos', name: 'consultar-articulos')]
public function consultarArticulos(
    ManagerRegistry $doctrine
): JsonResponse {
    $articulos = $doctrine->getRepository(Articulos::class)->findAll();
    $json = array();
    foreach ($articulos as $articulo) {
        $json[] = array(
            'id' => $articulo->getId(),
            'titulo' => $articulo->getTitulo(),
            'publicado' => $articulo->isPublicado(),
        );
    }

    return new JsonResponse($json);
}
```

- Lo visualizamos en el navegador, usando el JsonViewer de Chrome:
  - http://localhost:8000/consultar-articulos

4. Por último vamos a sacar SELECT * FROM articulos WHERE publicado=1 AND titulo = "Bienvenidos" usando como salida JSON y el método findBy():

- Creamos otro método en el mismo controlador:
  - En src/Controller/ArticulosController
- Añadimos el nuevo método:

```
#[Route('/articulos/{publicado}/{titulo}', name: 'ver-articulo2')]
public function verArticulo2(
    ManagerRegistry $doctrine,
    bool $publicado,
    String $titulo
): JsonResponse {

  // Dentro del findBy metemos 2 arrays
  // El 1er array es para filtrar por varios campos
  // El 2º array es para cambiar la ordenación
  // En este caso, la id irá al revés: 3,2,1...
    $articulos = $doctrine->getRepository(Articulos::class)->findBy(
        [
            'publicado' => $publicado,
            'titulo' => $titulo
        ],
        ['id' => 'DESC']
    );

    $json = array();
    foreach ($articulos as $articulo) {
```

```
            $json[] = array(
                'id' => $articulo->getId(),
                'titulo' => $articulo->getTitulo(),
                'publicado' => $articulo->isPublicado(),
            );
        }
        return new JsonResponse($json);
    }
```

- Lo visualizamos en el navegador, usando el JsonViewer de Chrome:
    - http://localhost:8000/articulos/1/Bienvenidos

> UN TRUCO!! Podemos poner lo de antes asi también:
> http://localhost/symfony6/public/index.php/articulos/1/Bienvenidos servidor - ruta_carpeta - endpoint

5. Las 3 consultas anteriores con salida en formato JSON lo podemos comprobar en mysql

```
mysql> SELECT * FROM articulos WHERE id = 1;
+----+----------+--------------------+-----------+
| id | autor_id | titulo             | publicado |
+----+----------+--------------------+-----------+
|  1 |        1 | Manual de Symfony5 |         1 |
+----+----------+--------------------+-----------+
1 row in set (0,00 sec)

mysql> SELECT * FROM articulos;
+----+----------+--------------------+-----------+
| id | autor_id | titulo             | publicado |
+----+----------+--------------------+-----------+
|  1 |        1 | Manual de Symfony5 |         1 |
|  2 |        1 | Notas sobre GIT    |         0 |
|  3 |        1 | Bienvenidos        |         1 |
|  4 |        1 | Manual de Symfony5 |         1 |
|  5 |        1 | Notas sobre GIT    |         0 |
|  6 |        1 | Bienvenidos        |         1 |
+----+----------+--------------------+-----------+
6 rows in set (0,00 sec)

mysql> SELECT * FROM articulos
    -> WHERE publicado=1 AND titulo = "Bienvenidos";
+----+----------+-------------+-----------+
| id | autor_id | titulo      | publicado |
+----+----------+-------------+-----------+
|  3 |        1 | Bienvenidos |         1 |
|  6 |        1 | Bienvenidos |         1 |
+----+----------+-------------+-----------+
2 rows in set (0,00 sec)
```

Consultar Objetos (Avanzado)

Tabla de contenidos

1. Vamos a crearnos un nuevo método en el controlador de Articulos:

```
#[Route('/ver-articulos-autores/{publicado}/{titulo}', name: 'ver-
articulos-autores')]
    public function consultarArticulos3(
        ManagerRegistry $doctrine,
        int $publicado,
        String $titulo
    ): JsonResponse {

      // En este caso no usamos el gestor de entidades
      // Usamos sólo la conexión
        $connection = $doctrine->getConnection();

        // IMPORTANTE: si queremos personalizar la salida
        // JSON, poner alias en los campos del SELECT
        $articulos = $connection
            ->prepare("SELECT articulos.id as articulos_id,
                       titulo, publicado,
                       nombre as Escritor
                       FROM articulos, autores
                       WHERE autor_id = autores.id
                       AND publicado = $publicado
                       AND titulo = '" . $titulo . "'")
            ->executeQuery()
            ->fetchAllAssociative();

        // Con el dump, sacamos el array completo
        return new JsonResponse(dump($articulos));
    }
```

2. Y lo probamos en el navegador

- http://localhost:8000/ver-articulos-autores/1/Bienvenidos

Actualizar Objetos

Tabla de contenidos

- Vamos a seguir con el CRUD. Ahora el UPDATE. En src/Controller/ArticulosController

1. Vamos a crearnos un nuevo método en el controlador de Articulos: Sacaremos el UPDATE articulos SET titulo="nuevo titulo" WHERE id=2;

```
#[Route('/cambia-articulo/{id}/{titulo}', name: 'actualizar-articulo')]
public function cambiarArticulo(ManagerRegistry $doctrine, int $id, String
$titulo): Response
{
    $entityManager = $doctrine->getManager();
    $articulo = $entityManager->getRepository(Articulos::class)->find($id);

    if (!$articulo) {
        throw $this->createNotFoundException(
            'Articulo NO existe con ID: ' . $id
        );
    }

    $articulo->setTitulo($titulo);
    $entityManager->flush();

    // Aprovechamos el método anterior para presentar el registro cambiado
    return $this->redirectToRoute('ver-articulo', [
            'id' => $articulo->getId()
    ]);
}
```

2. Sacamos en primer lugar el error, intentando cambiar un artículo que NO existe: http://localhost:8000/cambia-articulo/5/aloha
3. Y ahora probamos con uno que SI existe: http://localhost:8000/cambia-articulo/3/aloha

- Incluso podemos poner un título con varias palabras http://localhost:8000/cambia-articulo/3/Bievenidos%20a%20Symfony

Eliminar Objetos

Tabla de contenidos

- Vamos a seguir con el CRUD. Ahora el DELETE. En src/Controller/ArticulosController

1. . Vamos a crearnos un nuevo método en el controlador de Articulos: Sacaremos el DELETE FROM articulos WHERE id = 2

```
#[Route('/elimina-articulo/{id}', name: 'eliminar-articulo')]
public function eliminarArticulo(ManagerRegistry $doctrine, int $id):
Response
{
    $entityManager = $doctrine->getManager();
    $articulo = $entityManager->getRepository(Articulos::class)->find($id);

    if (!$articulo) {
        throw $this->createNotFoundException(
            'Articulo NO existe con ID: ' . $id
        );
    }

    $entityManager->remove($articulo);
    $entityManager->flush();

    return new Response("Articulo con ID " . $id . " eliminado!");
}
```

2. Y probamos en el navegador: http://localhost:8000/elimina-articulo/2 http://localhost:8000/consultar-articulos

5. Forms

Tabla de contenidos

- Recursos
    - https://diego.com.es/creacion-de-formularios-en-symfony
    - https://symfony.com/doc/current/forms.html
- Para crear formularios, lo primero es instalar el módulo correspondiente:

```
composer require symfony/form
```

El proceso completo será el siguiente:

1. Crear la entidad para gestionar el formulario. En nuestro caso, usaremos /src/Entity/Autores
2. Creamos el controlador para gestionar el formulario. En nuestro caso, usaremos /src/Controller/AutoresController que ya tenemos
3. Añadimos el siguiente método para visualizar la tabla autores:

```
use Symfony\Component\HttpFoundation\JsonResponse;

//...
// class AutoresController extends AbstractController
// ...

#[Route('/consultar-autores', name: 'consultar-autores')]
    public function consultarAutores(
        ManagerRegistry $doctrine
    ): JsonResponse {
        $autores = $doctrine->getRepository(Autores::class)->findAll();
        $json = array();
        foreach ($autores as $autor) {
            $json[] = array(
                'id' => $autor->getId(),
                'nombre' => $autor->getNombre(),
                'edad' => $autor->getEdad(),
            );
        }

        return new JsonResponse($json);
    }
```

4. Agregamos las siguientes clases adicionales al controlador: En src/Controller/AutoresController.php

```
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Form\Extension\Core\Type\NumberType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
```

4. Y creamos un método nuevo para definir el formulario: En src/Controller/AutoresController.php

```
#[Route('/nuevo-autor', name: 'nuevo-autor')]
public function nuevoAutor(
    Request $request,
    ManagerRegistry $doctrine
) {
    $autor = new Autores();
    $form = $this->createFormBuilder($autor)
        ->add('nombre', TextType::class)
        ->add('edad', NumberType::class)
        ->add(
            'Guardar',
            SubmitType::class,
            array('label' => 'Crear Autor')
        )
        ->getForm();

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        $autor = $form->getData();

        $em = $doctrine->getManager();
        $em->persist($autor);
        $em->flush();

        // Cuando mandamos el formulario, vemos la tabla en JSON
        return $this->redirectToRoute('consultar-autores');
    }

    return $this->render('autores/index.html.twig', array(
        'form' => $form->createView(),
    ));
}
```

5. Tenemos que modificar el twig: En templates/autores/index.html.twig

```
{% extends 'base.html.twig' %}

{% block title %}Formulario Autores
{% endblock %}
```

```
{% block body %}
    <style>
        .example-wrapper {
            margin: 1em auto;
            max-width: 800px;
            width: 95%;
            font: 18px / 1.5 sans-serif;
        }
        .example-wrapper code {
            background: #F5F5F5;
            padding: 2px 6px;
        }
    </style>

    <div class="example-wrapper">
        {{ form_start(form) }}
        {{ form_widget(form) }}
        {{ form_end(form) }}
    </div>
{% endblock %}
```

6. (Opcional) Si queremos añadir un campo booleano: Usamos el CheckBox

- Primero añadimos el campo booleano sexo a Autores

```
php bin/console make:entity
Class name of the entity to create or update (e.g. BraveElephant):
 > Autores

 Your entity already exists! So let's add some new fields!

 New property name (press <return> to stop adding fields):
 > sexo

 Field type (enter ? to see all types) [string]:
 > boolean

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Autores.ph
```

- Y ahora hacemos la migración

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

- Y agregamos el nuevo input al controlador: En src/Controller/AutoresController.php

```
// ...
//$form = $this->createFormBuilder($autor)
  ->add(
      'sexo',
      ChoiceType::class,
      [
          'choices' => [
              'mujer' => true,
              'hombre' => false,
          ],
          'expanded' => true
      ]
  )
  /*
  ->add(
      'Guardar',
      SubmitType::class,
      array('label' => 'Crear Autor')
  )
  ->getForm();
  */
```

7. (Opcional) Si queremos podemos crear un ENDPOINT para actualizar los datos de los autores En src/Controller/AutoresController.php

```
// ...
#[Route(
        '/cambia-autor/{sexo}/{id}',
        name: 'actualizar-autor'
    )]
public function cambiarAutor(
    ManagerRegistry $doctrine,
    int $sexo,
    int $id,
): Response {
    $entityManager = $doctrine->getManager();

    $autor =
        $entityManager->getRepository(Autores::class)->find($id);

    if (!$autor) {
        throw $this->createNotFoundException(
            'Autor NO existe'
        );
    }
    $autor->setSexo($sexo);
    $entityManager->flush();
    // Aprovechamos el método anterior para presentar el registro cambiado
    return $this->redirectToRoute('consultar-autores');
}
```

Formulario -> Pokemons

Tabla de contenidos

- En templates/pokemons/index.html.twig

```
{% block title %}
  Hello PokemonsController!
{% endblock %}
{% form_theme formulario 'bootstrap_5_layout.html.twig' %}

{% block body %}
  <!-- ... -->

  <div class="example-wrapper">
    <h1>Formulario</h1>
    {{ form(formulario) }}
  </div>
{% endblock %}
```

- En src/Controller/PokemonsController

```
#[Route('/formulario-pokemons', name: 'pokemon7')]
    public function formularioPokemons(
        CategoriasRepository $repoCategorias,
        PokemonsRepository $repoPokemons,
        TiposRepository $repoTipos,
        ManagerRegistry $doctrine,
        Request $request
    ): Response {
        $pokemon = new Pokemons();
        $formulario = $this->createFormBuilder($pokemon)
            // El formato de campo será...
            // ->add (variable, tipoCampo, etiqueta)
            // variable debe ser la misma de las entidades!!
            ->add('id', NumberType::class, ['label' => 'ID'])
            ->add('nombre', TextType::class, ['label' => 'Nombre Pokemon'])
            ->add('sexo', ChoiceType::class, [
                'choices'  => [
                    'Ninguno' => null,
                    'Femenino' => true,
                    'Masculino' => false,
                ],
            ])
            ->add('sexo2', ChoiceType::class, [
                'choices'  => [
                    'Ninguno' => null,
                    'Femenino' => true,
                    'Masculino' => false,
```

```
                ],
            ])
            ->add('evolucion', ChoiceType::class, [
                'choices'  => [
                    'Si' => true,
                    'No' => false,
                ],
            ])
            ->add('idTipo', EntityType::class, [
                'class' => Tipos::class,
                'placeholder' => 'Elije Tipo2',
                'choice_label' => 'tipo',
                'label' => 'Tipo1 Pokemon',
            ])
            ->add('idTipo2', EntityType::class, [
                'class' => Tipos::class,
                'required'   => false,
                'empty_data' => null,
                'placeholder' => 'Elije Tipo2',
                'choice_label' => 'tipo',
                'label' => 'Tipo2 Pokemon',
            ])
            ->add('idCategoria', EntityType::class, [
                'class' => Categorias::class,
                'choice_label' => 'categoria',
                'label' => 'Categoria Pokemon',
            ])
            ->add(
                'Guardar',          // Ojo, es una variable! No meter
caract especiales
                SubmitType::class,
                ['label' => 'Guardar Pokemon']
            )
            ->getForm();

        $formulario->handleRequest($request);
        if ($formulario->isSubmitted() && $formulario->isValid()) {
            $pokemon = $formulario->getData();
            $em = $doctrine->getManager();
            $em->persist($pokemon);
            $em->flush();
            // Cuando mandamos el formulario, vemos la tabla en JSON
            return $this->redirectToRoute('pokemon2');
        }

        return $this->render(
            "pokemons/index.html.twig",
            ["formulario" => $formulario->createView()]
        );
    }
```

6. GraphQL

Tabla de contenidos

Instalación

- Recursos
  - https://www.linkedin.com/pulse/graphql-server-symfony4-abhishek-mishra
  - https://betterprogramming.pub/graphql-api-symphony-mongodb-c866a79fdf48
  - https://latteandcode.medium.com/symfony-primeros-pasos-con-overblog-graphqlbundle-f4ef937c8fb

1. Creamos la rama y nos introducimos en ella

```
cd $HOMEPROJECTS/symfony5-manual
git branch 4-GraphQL
git checkout 4-GraphQL
git push --set-upstream origin 4-GraphQL
```

2. Entramos en el proyecto e instalamos el bundle de GraphQL

```
cd $HOMEPROJECTS/symfony5-manual
composer require overblog/graphql-bundle
```

> IMPORTANTE: Si nos sale en la instalación Do you want to execute this recipe?, pulsamos Y [INTRO]

3. Es muy recomendable instalar la interfaz gráfica para probar la API:

```
composer req --dev overblog/graphiql-bundle
symfony server:start
```

> NOTAS ADICIONALES Define your schema, read documentation https://github.com/overblog/GraphQLBundle/blob/master/docs/definitions/index.md If you want to see your dumped schema (really not necessary for bootstrap): run bin/console graphql:dump-schema

4. Podemos ver el editor de GrapQL llendo a esta dirección: http://127.0.0.1:8000/graphiql

5. Ahora debemos añadir un prefijo a nuestras rutas para graphQL:

- Nos vamos a config/routes/graphql.yaml y dejamos puesto esto:

```

```

```
overblog_graphql_endpoint:
  resource: "@OverblogGraphQLBundle/Resources/config/routing/graphql.yml"
  prefix: graphql
```

7. API REST

Tabla de contenidos

- https://www.itdo.com/blog/primeros-pasos-con-symfony-5-como-api-rest/
- https://soka.gitlab.io/angular/conceptos/http/consumir-servicios-api-rest/consumir-servicios-api-rest/
- https://codingpotions.com/angular-servicios-llamadas-http

1. Lo primero es instalar Postman

- https://bytexd.com/how-to-install-postman-on-ubuntu/

```
# Instalar Postman (ojo es 1 línea)
tar -C /tmp/ -xzf <(curl -L https://dl.pstmn.io/download/latest/linux64) &&
sudo mv /tmp/Postman /opt/

# Ejecutamos Postman
/opt/Postman/Postman
```

8. SELECT

Tabla de contenidos

- Vamos a poner en este tema TODOS los elementos SELECT que hemos visto:
- Resumen de comandos SQL
    - USE -> Entrar en la BBDD
    - LIMIT -> Limita el nº de registros de salida
    - WHERE -> Filtrado de registros
    - WHERE...IN -> Filtro por varios registros
    - ORDER BY ASC/DESC -> Ordenación de datos
    - DISTINCT -> Distingue entre valores iguales
    - AS -> Alias, cambiar el nombre del campo en la salida
    - OPERADORES -> Son !=, <, >, <=, =>
    - Funciones AGREGACIÓN
        - AVG -> Media
        - COUNT -> Conteo de registros
        - MAX -> Valor máximo
        - MIN -> Valor mínimo
    - GROUP BY -> Agrupar registros -> OJO! Si se pone con el ORDER BY, ponerlo antes
- JOIN -> Unir tablas
    - Se emplea WHERE tabla1.campoA = tabla2. campoB

```
 USE ine;       # Entrar en BBDD

 # LIMIT -> Limita el nº de registros de salida
 SELECT * FROM municipios LIMIT 5;

 # SELECT campo1, campo2,..., campoN
 SELECT municipio, codprovincia FROM municipios LIMIT 5;

 # WHERE ->  Filtrado
 SELECT municipio FROM municipios WHERE codprovincia = 41;

 # WHERE...IN -> Filtro por varios registros
SELECT municipio FROM municipios WHERE codprovincia IN(41,11,14);

# ORDER BY ASC/DESC -> Ordenación de datos
SELECT municipio, codprovincia
FROM municipios
WHERE codprovincia IN(41,11,14)
ORDER BY codprovincia DESC;


# DISTINCT -> Distingue entre valores iguales
```

```
SELECT dc FROM municipios LIMIT 10;
SELECT DISTINCT dc FROM municipios LIMIT 10;

SELECT codprovincia FROM municipios LIMIT 200;
SELECT DISTINCT codprovincia FROM municipios LIMIT 200;

# Alias y operadores
SELECT municipio AS Poblaciones FROM municipios LIMIT 5;

SELECT DISTINCT codprovincia AS Provincias
FROM municipios
WHERE codprovincia > 41
LIMIT 5;

# El campo FILTRO no tiene porqué coincidir con la salida
SELECT municipio AS Poblaciones
FROM municipios
WHERE codprovincia = 41
LIMIT 10;

# Ejemplo: Las 5 provincias anteriores a Sevilla
DESCRIBE provincias;
SELECT provincia AS Provincias
FROM provincias
WHERE codprovincia < 41
ORDER BY codprovincia DESC
LIMIT 5;

# Funciones AGREGACIÓN -> AVG y COUNT
# Dame el número de poblaciones de Sevilla
SELECT COUNT(municipio)
FROM municipios
WHERE codprovincia = 41;

# Suma de poblaciones de Sevilla, Cádiz y Córdoba
SELECT COUNT(municipio)
FROM municipios
WHERE codprovincia IN (41,11,14);

# La misma consulta usand puertas lógica (peor rendimiento)
SELECT COUNT(municipio)
FROM municipios
WHERE codprovincia = 41
OR codprovincia = 11
OR codprovincia = 14;

# Dame la ultima ID de municipios de ALAVA (902-Lantarón)
SELECT MAX(codmunicipio)
FROM municipios
WHERE codprovincia = 1;

SELECT MAX(codmunicipio)
FROM municipios
WHERE codprovincia = 41;
```

```
# Sacar nº de poblaciones separadas de Sevilla, Cádiz y Córdoba
SELECT COUNT(codprovincia) AS "Nº Poblaciones",
codprovincia AS "Cod Provincia"
FROM municipios
WHERE codprovincia IN(41,11,14)
GROUP BY codprovincia;


# Nº de poblaciones de otras provincias DISTINTAS a Sevilla
SELECT COUNT(codprovincia)
FROM municipios
WHERE codprovincia <> 41;


# Sacar el Modelo (diagrama) de la BBDD
# Database > Reverse Engineer (Ingenieria Inversa)

USE ine;
SELECT *
FROM municipios
WHERE municipio = "Marbella";

SELECT provincia FROM provincias;

# Sacar nº de poblaciones separadas de
# Sevilla, Cádiz y Córdoba
# poniendo los nombres de las provincias
# y la región a la que pertenecen
# UNIMOS LAS 3 TABLAS de la BBDD!!
# Esto se llama JOIN

SELECT COUNT(codmunicipio) AS "Nº Poblaciones",
provincias.provincia AS "Provincia",
ccaa.comunidad AS "Región"
FROM ccaa, provincias, municipios
WHERE ccaa.codauto = provincias.codauto
AND provincias.codprovincia = municipios.codprovincia
AND municipios.codprovincia IN(41,11,14)
GROUP BY municipios.codprovincia;
```

9. CodeAnyWhere

Tabla de contenidos

- https://codeanywhere.com/signin
    - Pulsar en Sign Up y rellenar el formulario
    - Ir a nuestro correo y verificar pulsando el enlace
    - Entramos en la página con Sign In. Vemos el Dashboard (Tablero)
    - PUlsar New Container. Elegimos PHP, nombre: symfony y [Create]
- Una vez abierto el contenedor instalamos Symfony CLI y creamos el proyecto

```
curl -1sLf 'https://dl.cloudsmith.io/public/symfony/stable/setup.deb.sh' |
sudo -E bash
sudo apt install symfony-cli
symfony
cd ~/workspace
symfony new symfony --version=5.4 --webapp
cd symfony
```

- Entramos en la carpeta del proyecto e instalamos las dependencias

```
cd symfony
composer require --dev symfony/maker-bundle
composer require twig
composer require annotations
composer require symfony/orm-pack
composer require symfony/form
```

- IMPORTANTE: EN el lado derecho de la pantalla sale la previsualziación de la página en el navegador. Hay que cambiar el puerto que viene por defecto 3000 a 8000
    - https://port-3000-symfony-diweb2022771506.preview.codeanywhere.com
    - https://port-8000-symfony-diweb2022771506.preview.codeanywhere.com
- MySQL dentro del contenedor

```
mysql -u root
```

10. ANEXO

Tabla de contenidos

- Os voy a mandar los métodos de CochesController con algunos comentarios de lo que hemos visto en clase:
- En primer lugar los use...

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

use App\Entity\Coches;
use Doctrine\Persistence\ManagerRegistry;
use App\Repository\CochesRepository;
use Symfony\Component\HttpFoundation\JsonResponse;

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Form\Extension\Core\Type\NumberType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
```

- Y luego los método para ACTUALIZAR, BORRAR, y el formulario...

```
#[Route(
        '/cambia-coche/{potencia}/{electrico}',
        name: 'cambia-coche'
    )]
    public function cambiaCoche(
        ManagerRegistry $doctrine,
        int $potencia,
        bool $electrico
    ): Response {
        $entityManager = $doctrine->getManager();
        $coches =
            $entityManager->getRepository(Coches::class)->findAll();

        /* OJO! Si vamos a añadir potencia
        a la que tuviera el coche previamente
        obtenemos el dato de la BBDD y le sumamos
        la nueva potencias*/
        foreach ($coches as $coche) {
            $potenciaPrevia = $coche->getPotencia();
            $coche->setPotencia($potenciaPrevia + $potencia);
            $coche->setElectrico($electrico);
```

```
            $entityManager->persist($coche);
            $entityManager->flush();
        }

        // Aprovechamos el método anterior para presentar el registro
cambiado
        return $this->redirectToRoute('ver-coches');
    }

    #[Route('/elimina-coche/{id}', name: 'elimina-coche')]
    public function eliminaCoche(
        ManagerRegistry $doctrine,
        int $id
    ): Response {
        $entityManager = $doctrine->getManager();
        $coche =
            $entityManager->getRepository(Coches::class)->find($id);
        if (!$coche) {
            throw $this->createNotFoundException(
                'Coche NO existe con ID: ' . $id
            );
        }
        $entityManager->remove($coche);
        $entityManager->flush();
        return $this->redirectToRoute('ver-coches');
    }

    #[Route(
        '/ver-coches-tipo/{tipo}',
        name: 'ver-coches-tipo'
    )]
    public function verCochesTipo(
        ManagerRegistry $doctrine,
        String $tipo
    ): JsonResponse {
        // En este caso no usamos el gestor de entidades
        // Usamos sólo la conexión
        $connection = $doctrine->getConnection();
        // IMPORTANTE: si queremos personalizar la salida
        // JSON, poner alias en los campos del SELECT
        // Mucho OJO! Esto puede entrar en el teórico
        $coches = $connection
            ->prepare("SELECT modelo, nombre
                    FROM coches, tipos
                    WHERE tipos.id = tipo_id
                    AND nombre = '" . $tipo . "'")
            ->executeQuery()
            ->fetchAllAssociative();
        // Con el dump, sacamos el array completo
        return new JsonResponse(dump($coches));
    }


    #[Route('/nuevo-coche', name: 'nuevo-coche')]
```

```
    public function nuevoCoche(
        Request $request,
        ManagerRegistry $doctrine
    ) {
        $coche = new Coches();
        $form = $this->createFormBuilder($coche)
            ->add('marca', TextType::class)
            ->add('modelo', TextType::class)
            ->add('potencia', NumberType::class)

            // Para añadir el radio...
            ->add(
                'electrico',
                ChoiceType::class,
                [
                    'choices' => [
                        'si' => true,
                        'no' => false,
                    ],
                    'expanded' => true
                ]
            )
            ->add(
                'Guardar',
                SubmitType::class,
                array('label' => 'Crear Coche')
            )
            ->getForm();
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            $coche = $form->getData();
            $em = $doctrine->getManager();
            $em->persist($coche);
            $em->flush();
            // Cuando mandamos el formulario, vemos la tabla en JSON
            return $this->redirectToRoute('ver-coches');
        }
        return $this->render('coches/index.html.twig', array(
            'form' => $form->createView(),
        ));
    }
```